ThomasCarstens format

1 contributor

Raw Blame

430 lines (357 sloc) 8.84 KB

```c
#include <stdio.h> //printf
#include <stdlib.h> //malloc
#include <unistd.h> //open write read close
#include <string.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h> //pour le threading (makefile a modifier)

#include <errno.h> //errno perro
//char buffr;
//char * fgets(char *restrict s, int n, FILE *restrict stream);

// SCHEDULER: pour creer une interface
#define PROCESS_THREAD int main() {
#define END_PROCESS }

// //DIFFERENCE STACK AND HEAP
// dans /proc/pid/ voir cat maps
#define STR_SIZE 30
#define ALLOCATE(SIZE) (malloc(SIZE))

int main(){
  int a;
  int *b = malloc(sizeof(int));
  printf("%i %p %p\n", getpid(), &a, b);
  scanf("%i", &a);
  return 0;
}


// PYRAMID: MY WAY
int main() {

  char branches=3;
  char asterisk=atoi(str);
  char bigasterisk=1;
  char row = 0;
  for (int row=0; row<atoi(str); row++){
    asterisk--;

    for (int i=0; i<asterisk; i++){
      printf("-");
    }
    for (int i=0; i<bigasterisk; i++){
      printf("*");
    }
    printf("\n");
    bigasterisk=bigasterisk+2;
  }

}

//WITH SCANF:///////////////

int pyramid() {

  char branches=3;
  char asterisk=atoi(str);
  char bigasterisk=1;
  char row = 0;
  for (int row=0; row<atoi(str); row++){
    asterisk--;

    for (int i=0; i<asterisk; i++){
      printf("-");
    }
    for (int i=0; i<bigasterisk; i++){
      printf("*");
    }
    printf("\n");
    bigasterisk=bigasterisk+2;
  }

  int read_integer(){
    int read_number;
    scanf ("%i", &read_number);
    return read_number;
  }
}
```

```c
  int main() {
    int taille = read_integer();
    pyramid(taille);
    return 0;
  }


//FGETS: GREGOR IMPLEMENTATION
// 1 lire une ligne de stdin
// a. Allouer la mémoire pour cette string
char *str = ALLOCATE(STR_SIZE * sizeof(char));
// b. Lecture au clavier
fgets(str, STR_SIZE, stdin);

// 2 l'afficher sur stdout
printf("%s\n", str);

return 0;
}

str buff[10];

  int *a = malloc(30*sizeof(int));
  //mettre les 30 int a 0
  //a[1]= 0;
  for (int i=0; i<30; i++){
    *(a+i)=i;
    //printf("%i\n", *a);
    sprintf(buff,"%*i\n", *a);
    if i=30 {
      a[i]='\0';
    }
  };
  printf(*a);
  free (a);
  return 0;


//FGETS AVEC INSPECTION
  char name[10]="a";
  // printf("Who are you? ");
  printf("%s", name);
  while (atoi(name)!=1){
    fgets(name,10,stdin);
  };
  printf("%s.n",name);
  return(0);


//POINTERS
  int a = 42;
  int *b = &a;
  printf("%p\n", &a);
  return 0;
  while(fgets(char buffr, 10 , stdin) != NULL)
   {
      printf("%s\n", buffr);
   }

void f(int *a){
  *a = 42;
}
// ecrire sur un pointeur
// ecrit au prealable moi meme
int main() {
  int a;
  f{&a);
  printf("")
}

//ALLOUER MANUELLEMENT MEMOIRE ET VOIE CA DANS /proc/pid/ cat maps
int main() {
  printf("%i %p\n", getpid(), sbrk(0));
  int a;
  scanf("%i", &a);
  return 0;
}

//Write a program that asks for numbers (integers), stores them in the heap and when
// no number is supplied, prints the total sum of what was inputted.
//NOT VERIFIED.
int main(){

  reps=5;
  for (int i=0; i<reps; i++){
    char *b = malloc(10 * sizeof(int));
    fgets(b[i], 10, stdin);
    printf("%i\n", *b[i]);
    sum += *b[i];
    printf(sum);
  }

}

//STRUCTS
//NOT VERIFIED.
struct Point {
  float x;
  float y;
}

int main() {
  struct Point a_point = {1.0, 1.0}; //dans ma stack
  printf("%f", a_point.x);
```

```c
  struct Point *b_point= malloc(sizeof(struct Point)); //dans ma heap
  printf("%f", b_point -> x);
  //meme chose que *(b_point).x
  return 0;
}


////////////////////////////////////////
// //STRUCT TYPEDEF
typedef struct
{
  int x;
  int y;
} the_structure;

int main(){
  the_structure a;
  a.x = 0.0f;
  return 0;
}



struct Point {
  float x;
  float y;
};

int main() {
  char name[10]="a";

  printf("vector components: \nx: ");
  fgets(name, 10, stdin);
  a_point.x = atoi(name);
  printf("\ny: ");
  fgets(name, 10, stdin);
  a_point.y = atoi(name);
  int centroid = (a_point.x+a_point.y)/2;
  printf("\ncentroid: %i", centroid);
  return 0;
  //CORRECTION: FOR MULTIPLE POINTS
  // find nb of points to input. i=nb_points
  for (int i=0; i<q; i++){
    float x, y;
    scanf("%f %f", &x, &y);
    struct Point my_point = {1.0, 1.0}; //dans ma stack
    numbers[i] = my_point;
    //then compute centroid
      for (int z=0;z<q;z++){
        sum_x += numbers[z].x;
        sum_y += numbers[z].y;
      }
  }
}

////////////////////////////////////////////////
char buf[128];
// //FILEMANIP with syscalls. Ca peut aussi
//fonctionner avec :strings vers l'internet"
// NOT VERIFIED
#define BUFFER_SIZE 1024
int main(int argc, char const *argv[]){ //argc=nb args
                                        //argv -> char* -> item
  char buffr[10]; //= malloc(50*sizeof(char))

  void *buffer = malloc(BUFFER_SIZE);
  printf ("%s\n", argv[1]);
  int fd = open (argv[1], O_RDONLY);

  //int fd = open("testFile", O_CREAT, O_RDWR);
  if (fd== -1) {
    perror("open");
    return EXIT_FAILURE;
  };

  char str[]= "Bonjour,";

while(1){
  ssize_t read;
  long bytes_read = read (fd, buffer, BUFFER_SIZE);
  if (bytes_read== -1) {
    perror("read");
    return EXIT_FAILURE;
  };

  if (bytes_read==0){
    break
  }

  long bytes_written = write(STDOUT_FILENO, buffer, bytes_read);//write(fd, str, 1);
  if (bytes_written== -1) {
    perror("write");
    return EXIT_FAILURE;
  };

  printf("fd: %d\n", fd);
}
  //printf("bits: %ld\n", read(fd, buffr, 50));
  //printf("buf: %s\n", buffr);
  close(fd);
  free (buffr);
}

////////////////////////////////////////////////
```

```c
//FILEMANIP with FILE* abstraction.
//https://www.tutorialspoint.com/c_standard_library/c_function_fread.htm
int main(){
  char buf[128];
  FILE *file=fopen("testFile.txt", "w+");
  //size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
  //fread(void *ptr, size_t size, 50, *file);

  char d[] ="Hello stream";
  //fwrite(const void *ptr, size_t size, size_t nmemb,
              //FILE *stream);
  fwrite(d, strlen(d)+1, 1, file);
  fseek(file, 0, SEEK_SET);
  /* Read and display data */
  fread(buf, strlen(d)+1, 1, file);
  printf("%s\n", buf);

  fclose(file);
}
//pointeur sur un tableau sur une chaine de characteres


//   //stat --format "%B" file;
//   //OUTPUT SIZE OF MY FILE
    //https://linuxhint.com/stat-system-call-linux/
int main(int argc, char const *argv[]){
  struct stat my_stat;

  if (stat(argv[1], &my_stat) == -1){
    perror("stat");
    return 1;
  }
  printf ("size: %ld\n", my_stat.st_size);
  printf ("stat fichier: %ld\n", my_stat);
  return 0;
}


////////////////////////////////////////////////
//THREADS
//GREGOR
void function(){
  printf("bonjour\n");
}

int main (int argc, char const *argv[]){
  //int (*f1_ptr) {long, int} = &f1;
  //(*f1_ptr)
  pthread_t thread1;
  pthread_create(&thread1, (NULL), (void * (*)*void *)) &function, NULL);
  pthread_join(thread1, NULL); //BLOQUANTE
  //non teste, mais possible de trigger avec scanf("%i", thread1)
  return 0;
}

// TXA IMPLEMENTATION
//makefile: gcc -pthread main.c -o main
// actually causes the linking
// to be done by the linker

//I just learned passing by ref only works in C++
//NOT FINISHED.
int counter;
pthread_mutex_t lock;

void* Thread1(int counter) {
        printf("Hello world from other thread!\n");
        pthread_mutex_lock(&lock);
        counter++;
        pthread_mutex_unlock(&lock);
        //printf("%d", counter++);
        int sleeptime =1;
        //sleep(sleeptime);
        return NULL;
}

void* Thread2(int counter) {
        printf("Hello world from other thread!\n");

        pthread_mutex_lock(&lock);
        counter++; //critical section.
        pthread_mutex_unlock(&lock);

        printf("%d", counter);
        int sleeptime =1;
        //sleep(sleeptime);
        return NULL;
}

int main() {

        pthread_mutex_init(&lock,NULL);

        pthread_t t1;
        pthread_t t2;
        printf("Spawning thread.\n");
        pthread_create(&t1, &counter, (void*)Thread1, NULL);
        pthread_create(&t2, &counter, (void*)Thread2, NULL);

        for (int i = 0; i < 100; ++i)
          {
            if (pthread_join(t1, NULL) != 0)
              {
                fprintf(stderr, "error: Cannot join thread # %d\n", i);
```

```
            }
          }
          for (int i = 0; i < 100; ++i)
            {
          if (pthread_join(t2, NULL) != 0)
            {
              fprintf(stderr, "error: Cannot join thread # %d\n", i);
            }
          pthread_mutex_destroy(&lock);
        }

        printf("Program end.\n");
        return 0;
}


////////////////////////////////////
// ERROR DETECTION
// int main(int argc, char const *argv[]){
//   char* str = malloc(50*sizeof(char));
//   if (str!=NULL){
//     // il y a eu erreure
//     perror("malloc");
//     return 1; //fonction indique donc l'erreure
//     //return EXIT_FAILURE; //alternatif
//   }
//   return 0;
// }
//apt-get install moreutils
//errno -l //POUR VOIR LES ERREURES PROGRAMME


// int main(){
//   printf("O_CREAT %i\n", O_CREAT);
//   open("fichier", O_CREAT | O_SYNC); //| veut dire ou bit a bit
// }
```